Reliability study of an embedded operating system for industrial applications

Pardo, J., Campelo, J.C, Serrano, J.J.

Fault Tolerant Systems Group
Polytechnic University of Valencia.
Camino de vera, s/n. 46022 Valencia, Spain.

juaparal@upvnet.upv.es, jcampelo@disca.upv.es

Abstract: Critical industrial applications or fault tolerant applications need for operating systems (OS) which guarantee a correct and safe behaviour despite the appearance of errors. In order to validate the behaviour of an operating system in front of errors, software fault injection techniques can be used. These techniques can be used to corrupt the information of some of the operating system calls to see how the system react in front of invalid or corrupted values at the kernel calls. The research work presented in this paper is about the development and results obtained from the experimentation on software fault injection in an embedded system composed by a Real-Time Operating System (RTOS) like MicroC/OS-II and a microcontroller as the Infineon C167. A software fault injection tool has been developed. The methodology proposed treated the operating system as a black-box where the source code was not available. With this objective a layer between the operating system and the application to be executed has been developed. OS error detection coverage has been measured and observations about OS critical data structures to be improved have been commented, in order to improve the final robustness of the operating system.

1 Introduction

Software of computer systems involves a lot of fields and aspects of our lives. Despite their enormous expansion today, they are still far from reaching the perfection. In order to measure the quality of the software some tests are required. Fault tolerance deals with software's ability to hide problems, specifically the effects of faults [Voas98].

Robustness is defined as the degree to which a system operates correctly in the presence of exceptional inputs or stressful environmental conditions. Robustness can thus be viewed as an indication on the OS capacity to resist/react to faults induced by the applications running on top of it, or originating from the hardware layer or from device drivers [DBench02].

The research work presented is about the development of a software fault injection tool for a commercial-off-the-shelf (COTS) real-time operating system like MicroC/OS-II in an embedded system with an Infineon C167 microcontroller, typical for the automotive industry. The objective is to evaluate the OS error detection coverage, error propagation and error detection latency times in a fast and easy way of prototyping. It could be very interesting to be able to obtain data about robustness and dependability of an OS, and thus for the future to be able to compare different COTS RTOS in similar hardware platforms.

2 System Design

The main motivation to use Commercial Off-The-Shelf (COTS) components in a system design is the notorious cost reduction associated to the final product development. The use of COTS components becomes a cost-effective method for rapid prototyping of complex software systems. On the other hand, the use of COTS software components have serious certification problems due to their design process is unknown. COTS software is composed of general purpose components which have poor dependability specifications. COTS components are usually like a black-box, the source code is not available and their internal architecture (structure and data flow) is not adequately documented.

The selection of MicroC/OS-II as COTS RTOS came motivated from the perspective that it is a system widely used since several years ago. The first MicroC/OS version appeared in 1992. Since then, it is possible to find different applications and systems like industrial robots, motor control, medical instruments, etc., which use this operating system. Moreover, MicroC/OS-II is 99% compliant with the Motor Industry Software Reliability Association (MISRA) C Coding Standards. These standards were created by MISRA to improve the reliability and predictability of C programs in critical automotive systems. On the other hand, all Modified Condition Decision Coverage (MCDC) code in MicroC/OS-II has been removed, improving code quality for RTCA / EUROCAE DO-178B Level A-certified environments for avionics applications [VS04]. About the microcontroller it has been used one of the Infineon C166 family, typical in automotive embedded systems.

The aim of the study was to use a black-box approach for the OS study, i.e. it was supposed that the OS source code wasn't available. So the OS source code was not modified trying to avoid as maximum as possible an intrusion in the OS normal behaviour. With this objective a layer had been developed between the OS and the industrial application to be executed. Through this layer the fault injection was realized in any of the parameters of the system calls to measure the OS robustness. The system workload was continuously running and consisted of a series of tasks executing the application. On the other hand, an injection agent developed was in charge of injecting faults and invalid values at the kernel calls in order to monitor the system robustness.

Relating to the software fault injection agent attributes, to denote that the main objectives of the injection were the fault prediction and OS robustness testing. Other objective was to check how the fault tolerance mechanisms, implemented in the OS and microcontroller, were activated. About the injection time, the injection was at runtime, selecting in a random way the system call, the parameter and the bit to be changed ('bit-flip'). Relating to the faults, injection level and localization was at a high level; due to the injections consisted of changing one bit in one of the parameters of the system calls. With this technique it was intended to check the behaviour of the detection and recovery mechanisms of the OS in front of errors.

The faultload is the most critical dimension of an OS benchmark and more generally of any dependability benchmark. Two techniques for system call parameter corruption can be used: the 'Bit-Flip technique', used in this work, consisting in flipping systematically bits of the target parameters and the 'selective substitution technique' when invalid data values are introduced in the system call parameters. Studies have demonstrated the equivalence of the errors provoked by the two techniques [Dbench02]. Finally, related to the persistence, the errors injected were of transitory type injecting one fault at a time.

3 Expected outcomes

Errors obtained after the fault injection could be grouped in two clearly differentiated groups; injected faults which no affected the system and injected faults which affected it. Injected faults producing errors on the system had been classified relating to the error detection mechanisms activated after the fault injection.

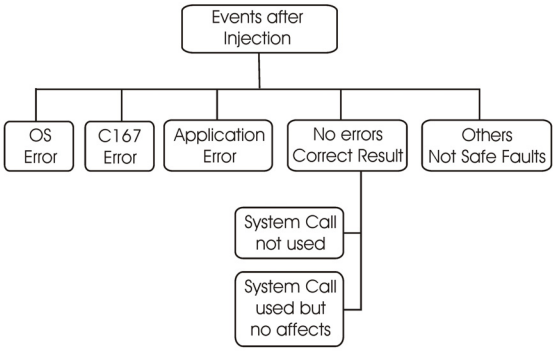

Fig. 1: Possible outcomes after the fault injection

As the figure 1 shows, after the injection of a fault several output results could be produced on the system:

•Operating system error code. The operating system detected something wrong and returned an OS error code at the kernel call execution.

•Infineon C167 error. The microcontroller produced an interrupt which aborted the application execution. An error through the exceptions handler like the stack overflow /underflow, undefined operation code, protection fault, illegal word operand access, etc. was produced.

•Application error. The obtained application execution result was different from the reference value, i.e. the expected execution value was incorrect.

•Nothing happened. This result meant the injected fault didn't affect the system. The injected fault was hidden by the system fault tolerance mechanisms. On the other hand, this situation could have been produced too, when the injection was realized in a system call which finally was not used, which it was not our case.

•And last situation, when the fault injection produced the system hang, this time the system didn't react. This situation had been solved through the use of the microcontroller watchdog timer to recover the system.

4 Analysis of the obtained results

Among the different variables stored in the output results file obtained, to highlight a column named as 'DETECC'. This column shows, in a codified way, which was the final service provided by the OS, i.e. which was the final result obtained after the injection of a fault during a supposed normal system working.

Codification of the different output values:

- D0: No error, correct output (the fault injection didn't affect the system).
- D1: Error detected by the operating system (µC/OS-II error code, 'OS Error').
- D2: Error detected by the application (the application result was no correct, 'Application Error').
- D3: Error which produced the system hangs ('Others, Not Safe Faults').
- D4: Error detected by the microcontroller ('C167 Error').

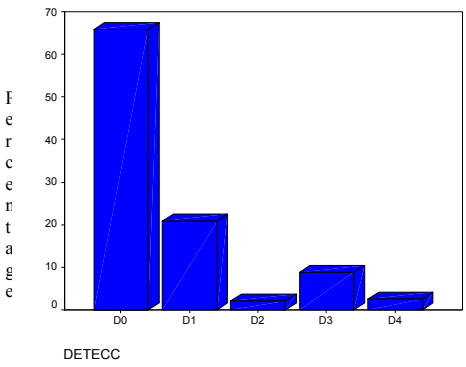

Fig. 2: 'DETECC' parameter results.

Seeing the figure 2, a high percentage (about 66%) of the faults injected in the parameters of the system calls didn't produce any failure on the system. The system through its fault tolerance mechanisms was able to hide the injected fault. The final OS error coverage obtained was:

$$C_{OS} = D0 + D1 = 86{,}7\ \%$$

The operating system coverage was given by those instances when the system finished the application execution in a correct way adding those instances when the MicroC/OS-II detected the error through its own detection mechanisms. So it could be said that the 86,7% of the times the system worked safe, i.e. the system was able to continue working in spite of the appearance of errors. Relating to the errors detection latency times, results were obtained in case of the output result parameter was 'D1'. These were the instances when the OS detected the error in one of its system calls. From the obtained statistics the error detection latency time was about 304 μs with a confidence interval from [265, 343], time taken by the system to detect the error.

5 Conclusions

After the realization of the experiments, the error detection coverage, error detection latency times, error propagation, typical OS error codes, etc. have been obtained. In a next research work, these data have to be compared with other operating systems working under the same conditions. Fault injection into the code and data memory segments of the microkernel will be implemented too. About possible improvements for the MicroC/OS-II to increase its dependability should take into account, that some detected errors in certain data structures could provoke critical failures on the system. These detected data structures should implement some mechanism to protect the information they host. To conclude for the future, it is necessary to test other different RTOS and COTS components, like OSEK or embedded RT-LINUX to have a criterion to be able to compare them.

References

[Arlat02] J.Arlat, JC. Fabre, M. Rodríguez, F. Salles. *"Dependability of COTS Microkernel-Based Systems"*. IEEE Transactions on computers, vol. 51, no.2. February 2002.

[DBench02] Pedro Gil , Jean Arlat , Henrique Madeira, Yves Crouzet, Tahar Jarboui, Karama Kanoun, Thomas Marteau, João Durães , Marco Vieira, Daniel Gil, Juan-Carlos Baraza, and Joaquín Gracia.'Fault Representativeness'. Deliverable ETIE2. Dbench European Project. Dependability Benchmarking (IST-2000-25425).

[Voas98] J. Voas, G. McGraw. *"Software Fault Injection: Inoculating programs against errors"*. Edit. Wiley. USA, 1998. ISBN: 0-471-18381-4.

[VS04] Validated Software Company. "*MicroC/OS-II MISRA C Compliance Matrix"* Application note AN-2004. http://www.validatedsoftware.com. Weston, Florida, October 23, 2002.